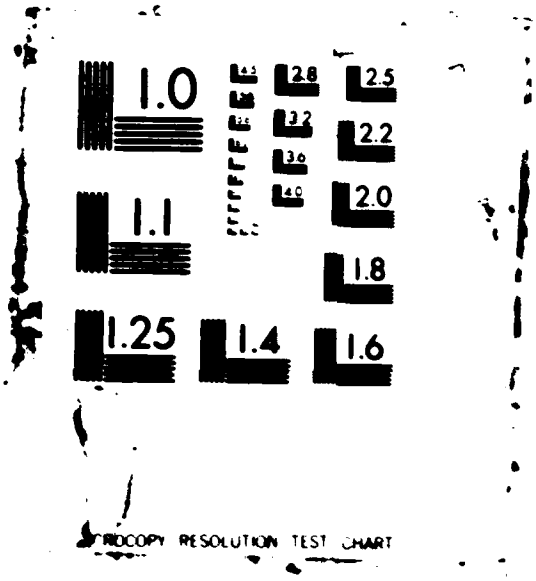
AD-A179 460 CONCURRENT COMPUTING: NUMERICAL ALGORITHMS AND SOME APPLICATIONS(U) MASSACHUSETTS INST OF TECH CAMBRIDGE V C KLEMA 15 JUL 86 AFOSR-TR-87-0408 AFOSR-82-0210 1/1

UNCLASSIFIED F/G 9/2 NL


1.0
1.1
1.25
1.4
1.6
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.8
ROCOPY RESOLUTION TEST CHART

Unc AD-A179 460

SECURITY CLASS

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION: Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release; distribution unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S): AFOSR-TR- 87-0408
6a. NAME OF PERFORMING ORGANIZATION: Massachusetts Inst. of Tech; 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION: AFOSR/NM
6c. ADDRESS (City, State and ZIP Code): 77 Massachusetts Avenue Cambridge, MA 02139	7b. ADDRESS (City, State and ZIP Code): Bldg 410 Bolling AFB DC 20332-6448
8a. NAME OF FUNDING/SPONSORING ORGANIZATION: AFOSR; 8b. OFFICE SYMBOL (If applicable): NM	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER: AFOSR-82-0210
8c. ADDRESS (City, State and ZIP Code): Bldg 410 Bolling AFB DC 20332-6448	10. SOURCE OF FUNDING NOS.

PROGRAM ELEMENT NO	PROJECT NO	TASK NO	WORK UNIT NO
61102F	2304	A3	

11. TITLE (Include Security Classification): Concurrent Computing: Numerical Algorithms and Some Applications

12. PERSONAL AUTHOR(S): Professor V. Klema

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Yr, Mo, Day)	15. PAGE COUNT
Annual	FROM _____ TO _____	July 15, 1986	6

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB GR	

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Finally, an important issue in harmonic retrieval problems, as in most computational problems, is the well-posedness of a particular problem instance. Specifically, when is a harmonic signal reconstruction by these methods particularly sensitive to additive noise? A definitive answer to this question was obtained in the form of a condition number for harmonic retrieval problems and is based on the Vandermonde determinant of the harmonic frequencies. This result was obtained via a combination of relationships using classical trigonometric moment theory and Toeplitz matrix conditioning. A fundamental new result, due to Davis and Bhatia, on the spectral sensitivity of unitary matrices was used. That result first appeared in the Princeton Conference on Information Systems and Sciences Proceedings in a paper titled "Conditioning of Eigenvector Methods for Beamforming Problems" while more recently a survey of this work will form the basis for an invited presentation at the upcoming IEEE Workshop on Spectrum Estimation. That survey is titled "The Sensitivity of Beamforming Problems".

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS ☐	Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL: Captain Thomas	22b. TELEPHONE NUMBER (Include Area Code): (202) 767-5026
22c. OFFICE SYMBOL: UNCLASSIFIED	

DD FORM 1473, 83 APR EDITION OF 1 JAN 73 IS OBSOLETE

AFOSR-TR- 87-0408

INTERIM SCIENTIFIC REPORT--AFOSR-82-0210-B, "Concurrent Computing: Numerical Algorithms and Some Applications," Principal Investigator, Virginia C. Klema.

This progress report is for the period July 15, 1985 through July 14, 1986.

Given the progress described in our previous reports, we began the current period with an operational, albeit primitive, version of the Software Tasker (Version 0) running on our small concurrent Intel systems. This software system provided a set of communication primitives that permitted the user to send code and data to worker processors, to initiate the concurrent environment, and to obtain status information on the application as it executed concurrently.

The first step in upgrading the system to Version 1.0 of the software required that the preliminary trap handlers for floating point arithmetic exceptions be incorporated in the Worker portion of the Tasker. The initial implementation of these handlers, described in our previous report, merely trapped the exception and aborted the job. Even though abortion of the offending task is often an appropriate action, these initial handlers did not provide enough information to enable the user to debug either his code or data. Therefore, before adding the exception handlers to the Worker operating system, we first needed to implement software to access the hardware on the Worker boards directly whenever an interrupt occurs -- thus allowing us to report additional diagnostic information to the user through the Manager. We then revised the trap handlers to include such information as an English language description of what occurred to trigger the exception, address information for both code and data, and identification of the offending operation and its parameters.

The remaining portion of the version 1.0 upgrade to the Tasker provides for badly needed traceback information to aid in the debugging process. Microprocessor systems, due to size limitations and the structure of their operating systems, do not maintain the data structures from which traceback information can be readily obtained. To the user, given only an address at which an exception occurs, this traceback information is essential; particularly since that address is more often than not deeply buried in one of the frequently called basic algorithmic models. Since the Tasker utilizes Intel's iRMX operating system kernel, the options for obtaining these data are somewhat limited. However, we have implemented a set of explicit primitives (notify_in and notify_out) which allow the user to request that traceback tables be built and reported to him in the event of an error -- whether on one of the Workers or the Manager.

Tasker Version 1.1 contains a number of upgrades to the existing system that have been designed but are not yet fully integrated into the software. The most important new features include: 1) the trap handlers that allow the user to take corrective measures and restart the offending computation rather than terminate the task, 2) a separate capability for logging information that adds time tags to user comments from either Worker or Manager and files them on the disk, and 3) the use of "ghost" processes that monitor both the hardware and selected software operations. Version 1.1 should be completed by Fall of this year.

In parallel with the development of the Tasker software, Richard Kefs has designed and implemented a new kernel operating system that is custom tailored to the concurrent computing environment. Based on the AMOS (A Minimal Operating System [Kr84]) design, this software provides the link between the hardware and a point just below the Tasker interface. Because of its design and the fact that it accesses the hardware directly, the potential exists for supporting more sophisticated diagnostic tables than is possible under the iRMX kernel -- particularly in the area of automatic trace back and usage

records. This work is documented in a master's thesis [Ke86], a copy of which was submitted to AFOSR.

Intel's iPSC system provides another environment for the development of concurrent applications. Delivered in mid July, the cube (designated so because of its hypercube architecture), presented a variety of software problems in its early days. Message collisions, lost messages on the nodes, system crashes and the long turn around time required to rebuild a core image, made the cube largely unusable by the project team until such time as Version 1.1 of Intel's operating system arrived in late December. This upgrade improved the development environment, however the message collision problem persisted. Largely as a defensive measure, motivated by the experience of having user processes die on individual nodes and losing even diagnostic information, we modified the design of the "ghost" process used in our Tasker (Version 1.1) and implemented it on the nodes.

From the perspective of the user, these three examples of concurrent operating systems share a common feature. Implementation of algorithms within these environments requires the creation of static load modules, with explicit communication requests and diagnostic commands. Although the Tasker interface provides services to the user at a higher level than the other two, all merely provide the user with the CAPABILITY for carrying out concurrent computing. None as yet provides the user with any assistance in defining his application in the first place or with the type of support for describing the decomposition of the algorithms or communications requirements of the problem.

A start has been made toward this objective in the form of a design for an application oriented interface layer between an operating system kernel running on the nodes and a set of specifications by which a user describes his application. This layer is intended to simplify the programming of the concurrent applications significantly, particularly by reducing the need to implement all communication paths and monitoring requests explicitly. This programmable environment for the execution of distributed tasks creates the link between an application and the minimal kernel. It supports the execution of the distributed application tasks while providing the following: 1) dynamic process-processor binding, 2) relative naming mechanisms inside the running task, 3) local trap handlers for floating point exceptions, 4) deadlock recognition, 5) error recovery, and 6) automated debugging mechanisms and performance analyzers.

The design calls for two distinct application entities: The first is the Application Code. The Application Code is a code fragment, or portion of an algorithm, written by the application programmer. This code fragment is combined with other code fragments and a set of instructions (or specifications) that indicate how these pieces are to communicate and be executed on different processors. The Application Code fragments plus specifications constitute the user's description of his distributed application task. The second application entity is the Application Manager, a unit of process control code which is associated with each Application Code fragment. This process control code (the programmable part of the operating system) is the software that services, monitors and debugs each piece of the user's distributed task during execution. Moreover, the Application Manager successively spawns additional Application Managers when needed to provide the connections between all Application Code fragments called out in the user's specifications until an entire application tree has been created. The Application Managers are written by the operating system programmer and are responsible for maintaining the consistency of the overall environment.

The approach being taken for the creation of the application interface described above is to work from the inside out -- stopping short of the mechanisms by which the user would

actually enter his specifications into the system. In working from the outside in, on the other hand, designing the input description and design tools for the user, a number of alternatives are being explored. One of the most promising paradigms at this time appears to be graphical -- using networks to represent the flow of data through the nodes (Application Code fragments). The initial design of a Concurrent Computing Graphics Support System calls for the design of this input management subsystem encompassing problem description, pre-execution analysis, and display, as part of Version 1.0 of the System. Design of the second component of the graphics system, or system management subsystem, will begin shortly thereafter. This second component receives the output from the Application Managers and generates analyses and display of both system and algorithmic efficiency; hence, it is also closely connected with the design of the programmable application interface described above. The third component, the output management subsystem, is the most problem dependent. Current plans call for the design of basic tools for displaying results which will be augmented as particular applications are tried.

During this same time period, we plan to establish a network linking the project's concurrent computing engines, graphics system, software development stations and file servers. We have a preliminary design for an application library that will permit network access and data transfer from inside individual processes. This would be in addition to the traditional file transfer capability needed to offload application development tasks from the single user concurrent machines. The application library as envisioned would be particularly valuable in passing much of the monitoring data to another node for analysis. Moreover, for appropriately designed applications, the network (through the application library) may serve as the communication media for a loosely coupled distributed system in which the nodes are the tightly coupled concurrent engines.

The time estimate for completing this network will be strongly dependent on how many of the appropriate hardware drivers and higher level network support routines are available commercially. We are in the process of assessing this availability.

The networking of our concurrent computing systems and the graphical display of the mapping of the application, debugging the computation and the communication, and monitoring the concurrent execution will be made possible by the recent award of a grant to Virginia Klema from the Department of Defense University Research Instrumentation Program for FY 86. This grant for research equipment includes networking components for linking the small concurrent Intel systems, the Intel cube, the VAX, and the associated workstations together with monitors and computing engines for visual display.

Our emphasis on research for the forthcoming year will be to focus on the software environment that supports numerical algorithms for scientific computation and its applications. The research is interdisciplinary work on numerical algorithms designed to take full advantage of IEEE floating point arithmetic as described in [AN85], and the operating system extensions to assist the user We will build on the experience gained during the past year, particularly the research, design, and implementation of the Software Tasker [Du86] and the minimal operating system to permit concurrent computing and support dynamic tasking for scientific applications [Ke86]

A major goal of the forthcoming research is to work on mapping a particular application from user-written specifications onto a concurrent computing system, debug the application program, and monitor its execution in real time. Such monitoring can aid the continuation of the application itself and, even more importantly, identify constraints that can be controlled to obtain portability among concurrent systems of different design Examples of concurrent systems with quite different design constraints are the distributed

systems such as the Intel hypercube with its point-to-point communication by message passing and private memory on each node, the BBN Butterfly with global memory available to each processing element, and the forthcoming IBM RP3 based on the Ultracomputer from Courant, a machine with massive global memory available to all processors.

In order to achieve this goal, certain research effort concentrates on a supporting software environment to assist the application programmer in the design and implementation of distributed algorithms. Explicitly, we are implementing a minimal node operating system to provide debugging support, monitor computation and communication, and display this information to the user.

The process of creating distributed algorithms and applications includes the design itself, coding, testing and monitoring, alternative configurations and evaluations. This entire process can be reduced significantly with an appropriate environment. The operating system will simplify the testing stage by providing a local environment, programmable at run-time, that monitors the execution of the distributed application depending on the level of debugging. Such monitoring includes communication, deadlock, synchronization of tasks, processors busy, and processors idle.

The operating system determines the mapping of the distributed application to available processing elements depending on the specifications of the application, and also establishes process-processor assignments and communication paths. Therefore the numerical analyst, algorithm builder, or application designer can evaluate correctness and efficiency of design and implementation by simply changing the associated specifications.

With such an operating system in place we can reduce significantly the amount of work associated with getting an application onto concurrent systems. For example, the actual coding need not include debugging statements (often new sources of errors). Location of pieces of code is transparent to the user, and data are sent to entities with relative identities, not to processors. Testing and reconfiguration are carried out without the need for recompilation which simplifies the evaluation process and the determination of, say, the optimum number of processors for the parallel execution. Furthermore, different executions of applications can co-exist in the same system.

To provide these features, an application is translated into an application tree in which the leaves correspond to the pieces of code to be executed by distinct processors. The operating system takes a set of specifications for the application from which it allocates processing elements, establishes communication paths, and sets debugging and monitoring levels. Monitoring of the application is carried out by the parent of each leaf. These parents are processes created by the operating system executing code belonging to the operating system.

At the present time the operating system is in its simulation phase on our VAX 11/730 with USD 4.2 Berkeley Unix. It consists of its kernel (memory, synchronization, process control, communication, time, and naming management) and outer layers. The outer layers manage application trees from a simplified set of specifications, creation, deletion, simulated processor and communication assignment, and paths. The outer layers also provide partial monitoring and heuristics on mapping. Further work will be done on debugging and monitoring levels, mapping, specification language, error detection and error recovery, and data acquisition. Following the completion of the simulation phase, the operating system will be transported to our small Intel tightly coupled systems and then to the Intel hypercube.

Along with the above work on operating system supporting environment, progress includes components of a node library of frequently used numerical algorithms for scientific computation. These components are designed to interface to the supporting environment in the sense that all exception detection and trap handlers are anticipated from the operating system side.

We emphasize that, up to the present time, our access to concurrent computing systems has been limited to Intel equipment. Fortuitously, we anticipate that in the near term, we will have access to the BBN Butterfly, and our software environments will migrate to that system. Personnel associated with the above research include Patrick Barton, Elizabeth Ducot, Richard Kefs, Virginia Klema and associated students. We are in the process of trying to obtain large scale applications as test vehicles for our software environments Professor Sanjoy Mitter has suggested an image processing application described in a recent Ph. D. thesis by Jose Marroquin, "Probabilistic Solution of Inverse Problems," which was supported partially by AFOSR Grant 82-0135B

Research in the area of eigenanalysis methods for harmonic retrieval has resulted in three significant results. The first deals with an efficient method for orthogonalizing highly structured Toeplitz data matrices. This algorithm is based on inner product computations and derives from techniques and ideas originally used by Kailath and his Stanford University group to solve so-called "close-to-Toeplitz" systems of equations arising in the theory of alpha-stationary processes. The major contribution of this method lies in the fact that it provides an order of magnitude less complexity for orthogonalizing large rectangular data matrices. A paper titled "Fast Orthogonalization of Toeplitz Matrices" that describes this method will appear in the SIAM Journal on Scientific and Statistical Computing in 1987.

The ability to orthogonalize such matrices then allows for the more efficient computation of singular value decompositions. This is due to the fact, first observed by T Chan (Yale University), that SVD's can be more efficiently computed when an efficient orthogonalization method is available as opposed to the classical bidiagonalization used in the Golub-Reinsch approach. Similar ideas are used in this work to simplify the singular value and hence eigenvalue computation. As above, an order of magnitude less work is therefore required in the computation of singular values of large rectangular data matrices. A paper describing these ideas titled "Fast Singular Value Computation of Structured Matrices" will appear in the IEEE Transactions on Acoustics, Speech and Signal Processing.

Finally, an important issue in harmonic retrieval problems, as in most computational problems, is the well-posedness of a particular problem instance Specifically, when is a harmonic signal reconstruction by these methods particularly sensitive to additive noise? A definitive answer to this question was obtained in the form of a condition number for harmonic retrieval problems and is based on the Vandermonde determinant of the harmonic frequencies. This result was obtained via a combination of relationships using classical trigonometric moment theory and Toeplitz matrix conditioning A fundamental new result, due to Davis and Bhatia, on the spectral sensitivity of unitary matrices was used. That result first appeared in the Princeton Conference on Information Systems and Sciences Proceedings in a paper titled "Conditioning of Eigenvector Methods for Beamforming Problems" while more recently a survey of this work will form the basis for an invited presentation at the upcoming IEEE Workshop on Spectrum Estimation That survey is titled "The Sensitivity of Beamforming Problems"

Cybenko recently discovered a fundamental formula describing the bahavior of Lanzos polynomials arising in the general Lanczos process for symmetric and unsymmetric

matrices. That result is one of the few available for the general case. It will appear in the special issue of Linear Algebra and its Application, dedicated to J. Wilkinson.

When one tries to parallelize any of these algorithms, a question that immediately arises is how to allocate subprocesses of a distributed solution to processors within a parallel machine so as to optimize some performance criterion. Our interest has been in harnessing the power of the hypercube interconnection network. An important question is thus whether a given dictributed algorithm can be mapped directly onto a hypercube in such a way as to preserve the locality of communicating processes. This problem was solved by David Krumme, N. Venkataraman (both of Tufts University) and G. Cybenko. The answer obtained was quite negative - not only is it not always possible to make such an allocation but the question of deciding whether an allocation exists was found to be NP-complete. Thus this problem is as difficult as classical intractable problems such as the Travelling Salesman Problem, integer linear programming and various scheduling problems. For practical purposes, any attempt at solving the problem will lead to an exponential growth (and hence infeasible) computation. The technique used to establish this result used a reduction of the boolean expression satisfiability problem to this hypercube embedding. That work has resulted in a number of publications. Two papers are currently being reviewed for publication in Information and Control, a third is submitted to Information Processing Letters while a fourth will appear in the First Hypercube Conference Proceedings. All papers are authored by Krumme, Cybenko and Venkataraman.

A variety of questions related to hypercube communication and allocation were studied. Some optimal routing algorithms were obtained and preliminary experiments with the use of simmulated annealing for allocation were done. That research is described in a series of publications by Krumme, Cybenko and Venkataraman. One will be presented at the upcoming International Conference on Parallel Computing, another will appear in the Proceedings of the Medium Scale Parallel Processors Workshop held at Stanford University in January 1986, while another paper will soon be submitted to the Journal of the ACM.

Work on a graphics system for displaying parallel computations was started on the INTEL iPSC hypercube. That work is the basis of Alva Couch's (a Tufts University Ph.D. student) thesis. The system allows users of both INTEL and NCUBE hypercubes to visualize parallel algorithm executions. That work will be described at the forthcoming Second Hypercube Conference in an invited presentation by Cybenko.

REFERENCES

[AN85] IEEE Standard for Binary Floating-Point Arithmetic, ANSI/IEEE Std 754-1985.

[Du86] E. R. Ducot, "Application Interface to the Concurrent Computing Environment," TR-86-02, Concurrent Computing Group, MIT Statistics Center, March, 1986.

[Ke86] R. Kefs, "Design and Implementation of a Minimal Multicoupled Operating System for Tightly Coupled Multiprocessors," Master's Thesis, Tufts University, and TR-86-01, Concurrent Computing Group, MIT Statistics Center, February, 1986.

[Kr84] D. W. Krumme, "Implementation of a Proposed Standard Real-time Operating System," Second IEEE Workshop of Real-time Operating Systems, Wakefield, MA, 1984.

END

5-87

DTIC